

Integer Overflow:

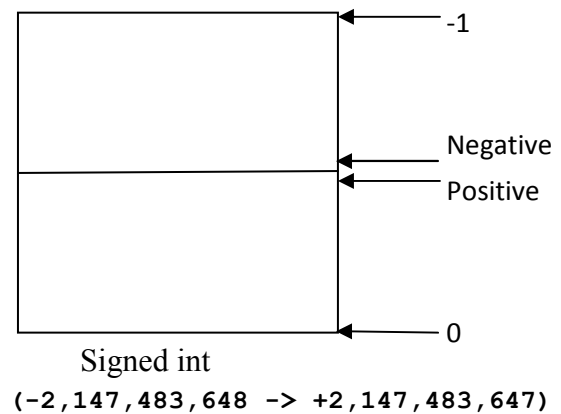
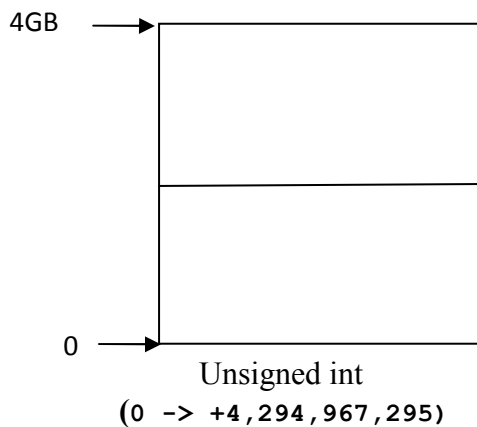
Example-1:

```
int x, y, z;  
x = 0x7fffffff;  
y = 1;  
z = x + y;  
printf ("%d", z);
```

Example-2:

```
unsigned int x;  
int y = -5;  
x = y;  
printf ("%d", x);
```

Unsigned int and signed int Ranges:



short to int Conversion:

```
int x;  
short int y;  
x = 100000;  
y = x;
```

"short int" range: -32,768 -> +32,767

"int" range: -2,147,483,648 -> +2,147,483,647

Security Problem Example:

```
unsigned int x;  
int y;  
read(net, &x, sizeof(x));  
y = x + sizeof_hdr;  
  
if( y < MAX_OFFSET)  
    Table [x] = 0;
```

Integer overflow here can lead to writing to a critical memory area.

Format String Bugs:

Various print functions:

- printf: prints to screen
- sprintf: prints to string
- snprintf: prints to string with bound

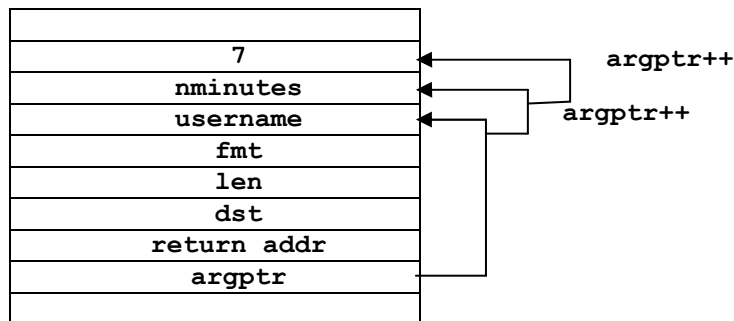
Example:

```
1) printf("Hello World!\n");  
2) snprintf(dst, sizeof(dst), "%s: %d minutes to %x\n", username, nminutes, 7);
```

```
int snprintf (char* dst, int len, char* fmt, ...)  
{  
    void *argptr = &fmt + 1;  
    .  
    .  
    .  
}
```

Stack:

```
snprintf(dst, sizeof(dst), "%s: %d minutes to %x\n", username, nminutes, 7);
```



As the format-specifiers are encountered, parameters are retrieved from the stack.

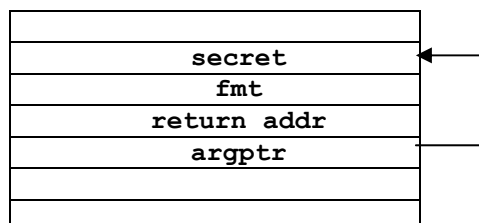
Format String Attack Examples:

Example-1: Too few parameters to printf function:

--Information leakage:

```
void caller(...)  
{  
    int secret;  
    printf("%d");  
}
```

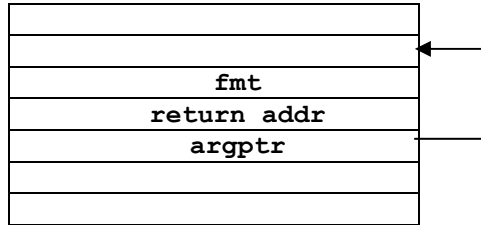
stack:



Example-2:

```
printf("%s", username); vs. printf(username);  
username: "%x%x%x%x...%x";
```

stack:

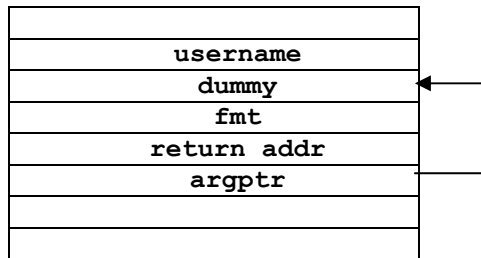


Thus, attacker can peek into the stack contents. Attacker can also figure out the exact location on the stack and may use it for the buffer overflow attack.

Example-3: Reading arbitrary memory location:

```
void caller(...)  
{  
    int dummy;  
    char username[512];  
    printf(username);  
    .  
    .  
    .  
}
```

stack:



```
username: "<0x12345678>%x%s"  
          ↑  
memory location that  
attacker wants to read
```

1. When %x is encountered, dummy is retrieved from stack.
2. When %s is encountered, contents of the memory location <0x12345678> are read because argptr points to username on stack.

username on stack:

